CHAPTER 11

Understanding Server Options

In This Chapter

The client-server model

A discussion of client processing

Web servers explored

A discussion of server processing using middleware

UNIX servers

NT servers

The longer you support and maintain a Web site, the more fancy things you want to do there. What kind of fancy things? You may include server variables, such as the date the page was last modified, the page's URL, or information stored in a database. Or, you may take information your visitors enter into your forms and use it to populate a database.

Before you can incorporate any of these variables, you need to understand the client-server model and processing options. This chapter gives you a solid understanding of what processing takes place where, what you must have installed to take advantage of server processing, what types of server processing software are available and how they work, and what you can take for granted on your server. This chapter also discusses peculiarities of both UNIX and NT servers.

The Client-Server Model

Lately, there's been a lot of press about the thin-client model. The *thin-client model* isn't a reference to a client of a modeling agency, but rather a reference to the idea of putting less computing power on the desktop and more on the server. Since the late 1980s, the debate has raged over where computer power should reside: on the desktop (thick client) or on the server (thin client).

Before the late 1980s, most people who did any computer processing had dumb terminals on their desktops, which were hooked up to some sort of a mainframe computer. The *dumb terminal* was only a screen, a keyboard, and a wire. The wire connected to the mainframe, where all the processing took place. The keyboard enabled you to enter data. The screen enabled you to see data. In the beginning, all the power resided at the hub, in a big, expensive mainframe computer. (See Figure 11-1 for an example.)

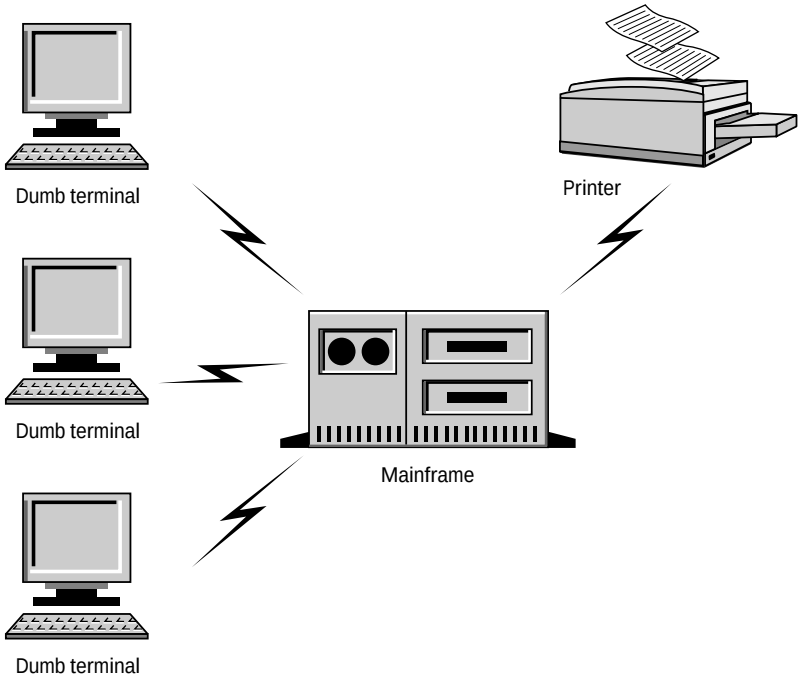

Figure 11-1: Dumb terminals connect to the mainframe at the center of the model

When personal computers (PCs) became affordable and useful software for PCs became available, businesses moved more processing from the center (the mainframe or server) to the periphery (the client). By the late 1990s, many offices had incredibly powerful computers on every desktop, far in excess of the power needed to do the kind of processing computer users were actually doing. This was the thick-client model, as shown in Figure 11-2. Because middle-sized servers have become so powerful for far less than the cost of a mainframe, the server has replaced the mainframe at the model's center.

The client-server model

The pendulum in business today is swinging back in the other direction to the idea of putting less power on the desktop and more on the Web server. Why? Several reasons:

- The growth of intranets, wherein many applications are delivered to the desktop via Web servers
- The continued need for shared files (and, thus, associated file servers)

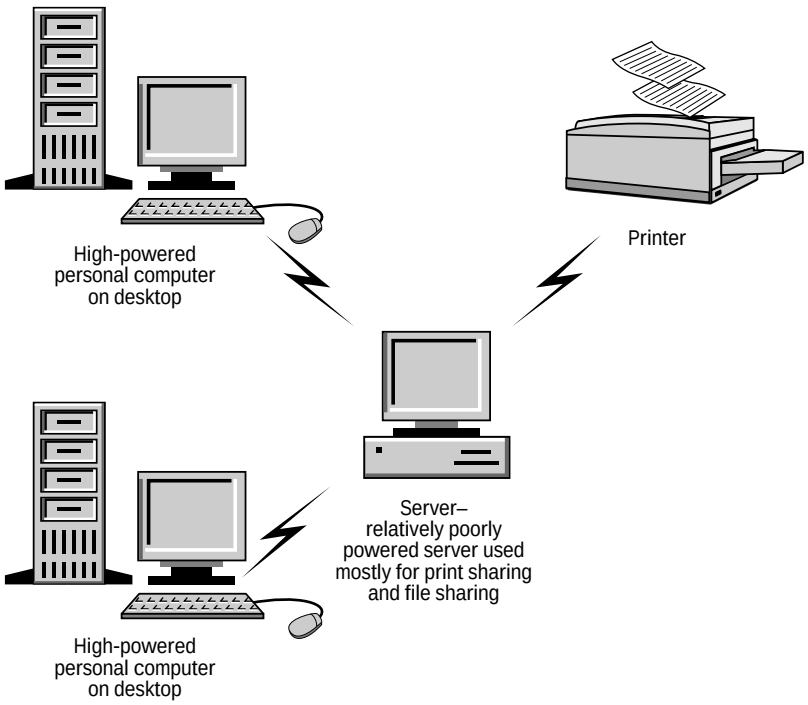

Figure 11-2: The thick-client model with a lot of power on the desktop

- The realization that for most office applications, desktops don't need the processing power of the latest, greatest personal computers
- The availability of client-server software engineered to run from a server

Processing: The crux of the issue

When you connect your computer to a server, where does the processing take place: on your desktop computer or on the server? That depends. When you request a Web page, if any processing is required other than serving the page, the processing usually takes place on the server. When you sign in to your Web server to publish your files (using the FTP client explained in Chapter 4), the processing takes place on your desktop computer.

Cleverly designed Web applications can move some of the processing burden from the server to the client. Remember, in most Web applications, as the Webmaster, you are paying for the cost of processing on the server, but not on the client. The client computer belongs to the person visiting your site. The way to get the most bang for your Web-page buck is to move any processing you can from *your* server to *the visitor's* client workstation.

Client Processing

So how do you take advantage of client processing? First you must understand what you can and can't do using client processing, as detailed in Table 11-1.

Table 11-1
Client Processing Capabilities and Restrictions

What You Can Do	*What You Can't Do*
Validate data entry. If you have form fields on your page, you can make sure they contain the right kind of data before re-submitting them to the server. How does this save server processing? First, your server doesn't have to do the validation. Second, data will only be submitted once, when it is complete, saving your server the trouble of having to serve the same page (with error messages on it) multiple times to the same client.	Serve a page. You can't serve a page using client processing. You can load another page that has already been downloaded with the current page, which can be a clever technique, but at some point all the data you display must come from the server (except for data collected from the visitor and data collected from the client workstation).
Rearrange data on the page. This is new to HTML 4! If you are running a page that serves data from a database, you can save your server the trouble of having to serve the same data multiple times, sorted differently, by taking advantage of client-side processing. The Document Object Model and client-side scripting enable you to deliver all the data to the client one time and then give the client the option of seeing it different ways. Visitors don't need to know that instead of going out to your server every time they want to see the data rearranged, the processing takes place on their machine. This can take a real load off your server.	Deliver data from a database. The database has to reside on a server (either your own or another you have permission to read). The only way to get this data to the client is to deliver it from a server.
	Collect data into a database. You have to get the data from the client's workstation back to the server. Client processing can't do this. The chapter on external databases (Chapter 45) covers this in detail.
	Read data from the visitor's computer. With few exceptions, you can't use client-side scripting to collect information about visitors or data from their machines. This kind of privacy safeguard for the Web surfer has contributed to the popularity and acceptability of client-side scripting. Other less innocuous technologies, such as ActiveX controls and Java applets, have failed to catch on as fast because they didn't safeguard the client workstation as well.

What You Can Do	*What You Can't Do*
Date and time processing. Say you need to create a clock in a Web page. No good reason exists to run back to the server at set intervals when every client computer has a built-in clock. With client-side scripting, you can get at the local client variables, including the date and time, and process this data on the client side.	Write data to a client workstation. With the exception of cookies, which are covered later in this chapter, you can't write anything to your visitor's computer.
Measure visitor response. If you want to do any kind of testing of visitor response, you can administer the entire test in one or more pages, without returning to the server, until the test is complete. This includes dynamic testing, where the next question asked is based on the answer to the current question. For testing the time it takes for visitors to complete questions, nothing will be more accurate than client-side processing.	

So how do you take advantage of client-side processing? Whenever you need to validate forms data, rearrange data on a page, perform date and time processing, or measure visitor response, you include a client-side script that does this using client power, rather than server power.

Chapter 48 teaches you what you need to know about JavaScript, the most popular language for client-side processing, to do all these things.

Introducing Web Servers

What exactly is a Web server? A *Web server* is a patient program that sits on your server (that is, the physical machine dedicated to serving pages and performing other server functions) waiting to receive an HTTP request via TCP/IP. Helpful, huh?

Any server configured to handle communications via TCP/IP (the Internet's communications protocol) has ports. These aren't physical ports, like the serial port and parallel ports on the back of your computer, but they serve the same purpose. All HTTP requests come through port 80 unless the server has been configured

differently. *Port 80* is the default Web server port. This is how your server, which may be a file server, an applications server, and an FTP server, in addition to being a Web server, keeps it all straight.

When an HTTP request comes through port 80 to the Web server, the Web server finds the page requested, checks the permissions of the client making the request, and, if the client has the appropriate permission, serves the page. Permission? Aren't all requests for pages anonymous? Not exactly. Figure 11-3 illustrates the request process.

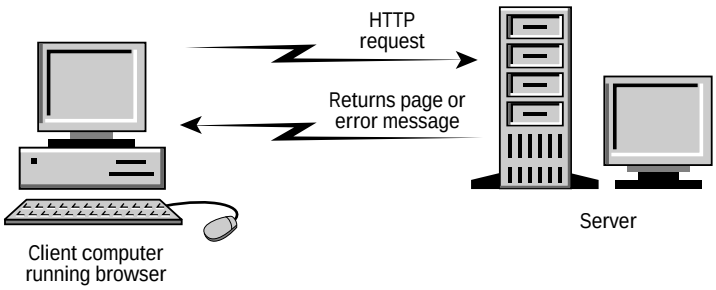

Figure 11-3: The client requests the page. Then the server evaluates the request and serves the page or an error message.

Permissions

Generally, HTTP requests are anonymous. What this really means is an account has been created on the Web server for HTTP requests. When a request comes through port 80, it is assumed to come from this account. Each file on the Web server has certain permissions associated with it. If the HTTP account (which may, in fact, be called *anonymous*) has adequate permission to read (or execute) that page, the Web server will serve that page.

All files and folders on a server have permissions associated with them. Whether or not you set them explicitly, they are there. Usually, when you create files on a server, they inherit the permissions of the directory in which you place them. The systems administrator usually sets the permissions for your directories. If you are having problems getting your page to load or seeing images on your page, have the systems administrator check the permissions on your files and directories to make sure they are, in fact, universally readable and executable.

NT: Hidden permission

What if you are sitting at a non-Windows-98/NT client? Then, if you try to get to a page that isn't accessible to the anonymous HTTP account, you either get an error message indicating you don't have permission to get to the page or a dialog box asking you to sign in.

Is this a good thing? It is a great thing for intranets! This means you don't have to rekey your ID and password into your browser every time you want to see a secure page. It can be a headache for Web server administrators because that much more security information must be monitored. For the Webmaster, it means more testing is required. You must test your pages from both Windows 98 and NT clients and others, signed in as someone with less authority than yourself.

NT servers handle permissions a little differently. Really, it's sort of sneaky. If you are sitting at a Windows 98 or Windows NT client and you request a page from a Windows NT Web server, you won't necessarily be making the request under the guise of the anonymous account set up for HTTP requests. In fact, your browser has sent your login ID and password (encrypted, of course) to the Web server with your request. If your ID is a valid ID on the Web server, then you are permitted to see any files on the server you have read (or executed). You are also able to see all the files the anonymous account is permitted to see.

Server Processing

What kinds of processing are best performed on the server? Any kind of processing or interaction with databases or any data stored on the server. How does it work? The visitor requests a page from the Web server. The Web server knows the page contains something it can't process and hands off the page to the appropriate auxiliary software for processing. The software that performs auxiliary processing, which sometimes comes installed with the Web server, converts the page into HTML and hands it back to the Web server. The Web server serves the page. Figure 11-4 illustrates this concept.

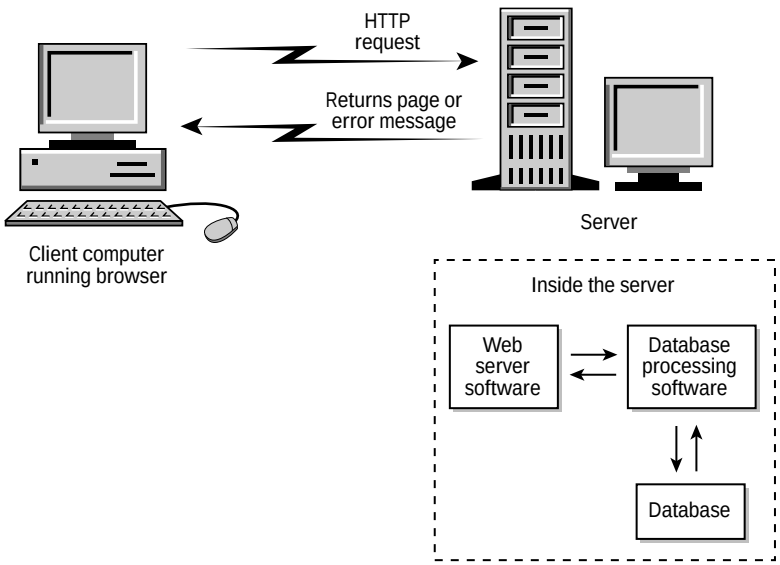

Figure 11-4: Server processing

What the Web server can't process

What can't the Web server process? Scripts, for one thing. Many proprietary flavors of server-side scripting exist. When a Web server comes across scripting that it recognizes isn't client-side scripting, it passes the page to the appropriate external software, which may be a middleware engine. Most often, pages with scripting not meant to be processed either by the client or by the server have different extensions. (Some you may have run across are in Table 11-2 later in this chapter.) Other times, the page has an extension of `.html`, but has a call to a script within the page. The Web server knows to hand this page to external software or a middleware engine (or has the processing capabilities built-in) when it comes to this code.

Middleware. Software that sits outside the Web server software, but on the same physical server, which performs specialized processing related to the Web, but also related to other technologies, such as databases.

Converting scripts to HTML

How does the middleware or external software produce an HTML page? The reason server-side processing is necessary is for a page to be able to pull data out of a database and present it on a page or to write data taken from a form to a database. Usually when you write a page using a scripting language, you indicate all records that meet certain criteria should be written into the HTML and served to the client. The middleware or external software goes out to the database (usually using ODBC standards), reads the database, and formats the data with HTML. Then it returns the page to the Web server. The Web server recognizes the page as containing only HTML and serves it to the client.

CGI script. The external software used to produce an HTML page is usually a Common Gateway Interface script written in C, C++, or Perl. The compilers for these scripts are usually free. The CGI script does the same thing as the middleware, but usually requires a higher level of technical proficiency to accomplish it.

The History of Middleware

Up until about 1995, a standard for software that performed auxiliary processing existed. It was called the *Common Gateway Interface* (CGI). Every type of server-side processing required the Webmaster to write a script in Perl, C, or C++. The script was saved with an extension of `.cgi`. When the Web server saw this extension, it read only the first line of the file and passed the script on to the appropriate compiler. Advantages to this system were: All three of these languages were commonly found on UNIX, most Web servers up until 1995 were UNIX based, and compilers for all three of these languages were in the public domain and likely to be found on your UNIX server.

The disadvantages were and still are as follows:

- ✦ All three of these languages are programming languages
- ✦ A single page of HTML may take pages of code to generate
- ✦ Making a change to the HTML on a page requires a programmer
- ✦ Hours of programming produce relatively modest results

Today, many people who use CGI programs either use pre-existing programs and don't make any programming changes to them, pay their ISPs to make programming changes, or are hardcore programmers.

Unfortunately, no standard exists in middleware. Quite a number of different types of systems are available to you as a Web developer to produce high-quality server-side processing. Most middleware revolves around interacting with a database. The advantages of using modern middleware include fast development time and the availability of rapid-application development (RAD) tools for many of these products. The disadvantage is cost. Most of them are commercial software. If you administer your own server, you can choose which one you like best. If you are already using an ISP with which you are happy, you will probably use one of the RAD tools it provides. Table 11-2 shows common extensions used to indicate that external processing is required.

Chapter 8 compares HTML-development tools, some of which are RAD tools for middleware.

Table 11-2
Extensions of Specific Languages

Extension	*Language*
.cgi	Common Gateway Interface (usually means the entire page is written in Perl and the result of the Perl script is a page of HTML)
.asp	Active server pages (proprietary to Microsoft)
.qry	Query page (proprietary to Tango)
.cfm	Cold Fusion markup language (proprietary to Allaire)
.js	Server-side JavaScript (used by Netscape servers for server-side processing)
.jsp	Java server pages (proprietary to Sun)

Rapid application development (RAD) tools are productivity-enhancing if they permit

- ✦ drag-and-drop access to data sources
- ✦ uniform-style application across all pages in your site or project
- ✦ multifile search-and-replace
- ✦ spelling check
- ✦ looking up of HTML tags
- ✦ auto-tag insertion
- ✦ wizards

and all without creating messy code.

Built-in middleware

The most popular type of server-side script processing comes built-in to the Web server. Two of the most popular commercial Web servers — Microsoft Internet Information Server and Netscape servers — use this approach. Internet Information Server (IIS) supports active server pages internally; Netscape servers support server-side JavaScript.

Built-in server-side script processing makes processing scripts transparent. For the server administrator, there is no additional software to install and configure. For the Web page developer, once you know what Web server software is running, you know what type of scripting you can use and you know it will work (as shown in Figure 11-5).

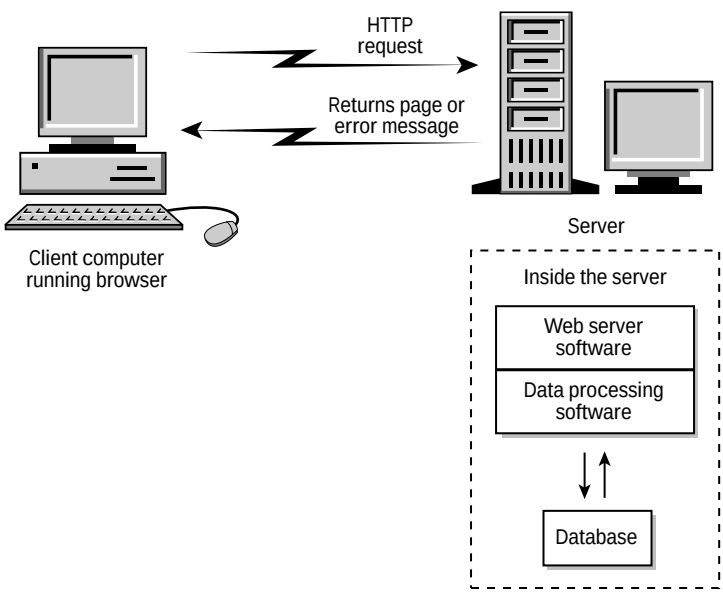

Figure 11-5: Built-in server-side script processing is transparent.

Cold Fusion

Cold Fusion by Allaire is one product that greatly increases what your Web site can do without requiring any programming. Using a simple language, called *Cold Fusion Markup Language* (CFML), you can create powerful scripts you write right into your HTML pages. The Cold Fusion server returns the script's results right into your page.

Some of the cool things you can do are as follows:

- ✦ Schedule the generation of a page daily, hourly, or at whatever interval you choose.
- ✦ Pull content off other sites and parse it into your own format. (Get permission from the site owner before you try this.)
- ✦ Send mail to everyone in a database from a Web page based on criteria indicated on the form on the Web page.
- ✦ Insert records into a database. Update a database record. Read a database for records that meet certain criteria.

Cold Fusion is available for both NT and UNIX (Solaris) platforms. It works with ODBC-compliant databases. You can find out more from the Allaire Web site (`www.allaire.com`).

Stand-alone middleware

The alternative to built-in server-side script processing is stand-alone middleware. Why use stand-alone middleware? You may be running a Web server that doesn't include server-side scripting (other than CGI scripts). Another good reason exists. Some stand-alone middleware expands the processing power of your Web server to such a great extent, it is worth investing in another software program to get these extra benefits. What might these capabilities include? Sending e-mail from the Web site using data in a database, for one. See the following sidebar for more information.

Rapid application development

The difference between easy-to-write and pain-to-maintain scripts is often the existence of an effective rapid application development environment. What is this? Essentially *rapid application development* (RAD) is a software program that makes it easy for you to write scripts. Microsoft includes support for active server pages in FrontPage. Netscape has a visual development environment for JavaScript. Allaire has Cold Fusion Studio for easy creation of Cold Fusion scripts.

Cookies

Cookies are small text files written to visitors' computers when they visit certain sites. These files enable the site to recognize visitors upon returning. Sometimes visitors want to be recognized and sometimes they don't.

As a Web developer, you can use cookies to identify visitors to your site. What can you do with this information?

- ✦ You can keep track of returning visitors.
- ✦ You can count new visitors.
- ✦ You can store visitors' preferences so they see what they told you they'd like to see.

Before you make the decision to use cookies, think about what kind of processing you want to do with this information. Writing a cookie is no work at all. Using the cookie effectively requires a lot more work.

The Web is a stateless system. A Web server doesn't know whether the same person has come to the Welcome page 100 times in a day or whether 100 different people have come once. This is why hit counts are so inaccurate. Cookies can change the stateless nature of the Web. If you know what you are doing with cookies, you can begin to collect valuable information about visitors to your site.

Cookies can also be abused. Many sites place cookies on visitors' pages and then do nothing with the data they are able to collect. Don't set cookies just because you can. You also need to remember visitors to your site have the option not to accept your cookies. You must make your site accessible even to people who don't want your cookies or can't save the cookie file (such as users at library computers or school computers). Figure 11-6 shows a typical cookie file.

Secure Servers

If you want to collect private information about your visitors or conduct any type of commerce, you need to set up a secure server. The term secure server is actually a misnomer. What is really secure on a *secure server* is the transmission of information, but because secure server is the expression everyone else uses, this book also uses this terminology.

How does a secure server work? When you go to a secure Web page, you will notice the lock on your browser (either in the lower left-hand corner of your screen on Netscape — or on the bottom toward the right part of your screen in Internet Explorer) is locked. By default, your browser gives you a warning every time you go to (or leave) a secure page, but you can turn off those annoying warnings (you may have already done this).

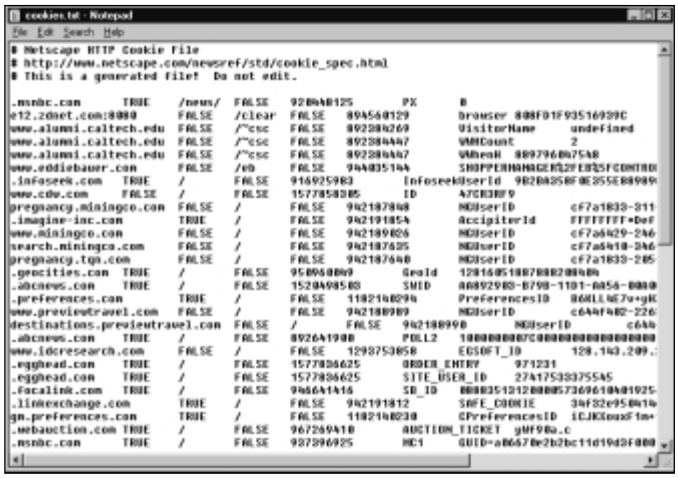
cookies.txt - Notepad
File Edit Search Help
Netscape HTTP Cookie File
http://www.netscape.com/newsref/std/cookie_spec.html
This is a generated file! Do not edit.

Figure 11-6: A cookie file

Normally, only pages that show you confidential information (such as the value of your mutual fund portfolio) or pages into which you enter confidential information (such as your credit card number) are secure. When this page was sent to your browser, it was sent encrypted. This means people with a lot of time on their hands who were sniffing packets on the Internet couldn't open your packets to read what was sent. When you enter information on a secure form, it is, likewise, sent encrypted back to the server. The term *secure server* says nothing about how tight the security is on the server itself.

A secure key is identical to a digital server certificate. The certificate provider, Verisign for example, assures end-users of whom they are communicating with.

Four steps are involved in setting up a secure server:

1. Get a secure key. The secure key is what your server uses to send encrypted data. You can get a secure key from Verisign (`www.verisign.com`).
2. Install the key on your Web server. Your server will have a key manager (as shown in Figure 11-7) or a flat file, where you install the key you purchase from the certification authority.
3. Publish your pages to the secure directory. Your systems administrator will be able to tell you where the secure directory is on your server. Any pages that need to be secure should be published to that directory.
4. Link to your secure pages using `https://` instead of `http://`. When you link to pages in your secure directory, instead of using relative links, such as `commerce/buynow.html`, you need to use absolute links, such as `https://www.sellingstuff.com/commerce/buynow.html`. Notice this absolute link uses the `https` protocol rather than the `http` protocol.

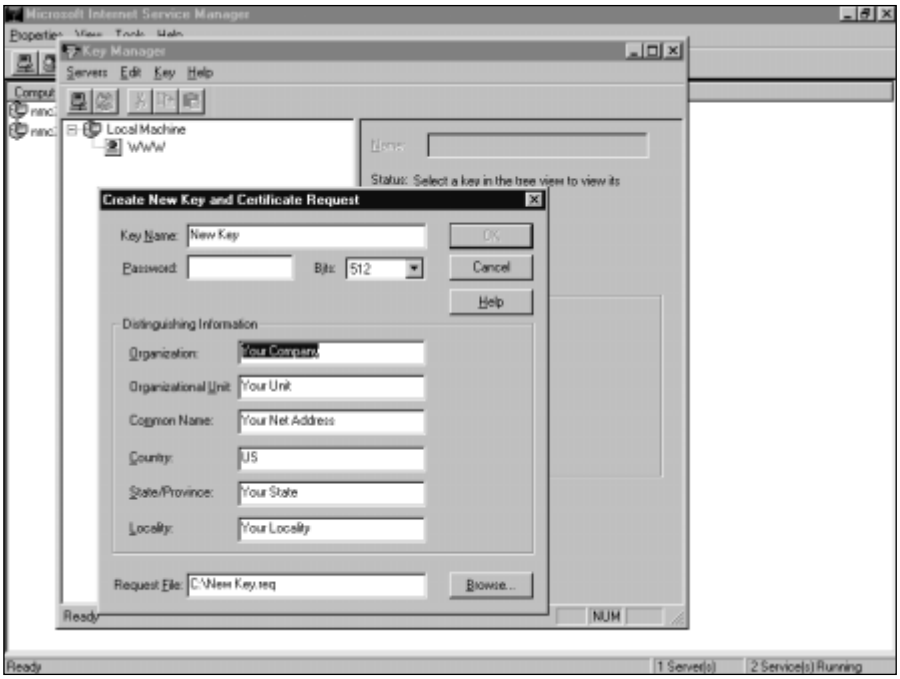

Figure 11-7: IIS Key Manager

Definition

Secure servers versus secure directories. Whether you will be required to publish all your secure pages in one directory or whether they can be anywhere on your server is really a policy issue. Just as with managing your media files, you may find placing all your secure pages (pages with which you communicate using the https protocol) in a single directory more convenient.

Looking at UNIX Servers

UNIX servers are the old warhorses of the Web server world. UNIX servers make reliable Web servers. The only drawback to UNIX Web servers is the environment isn't as graphical as it is with NT servers (or Mac servers, for that matter). More and more tools have become available to make publishing to a UNIX server more straightforward, such as the FTP clients shown in Chapter 4.

If you are publishing to a UNIX server, you must be aware of the following things:

- **Capitalization matters.** When you link from one page to another or when you publish a page, `MyPage.html`, `mypage.html`, `MYPAGE.html`, `MyPage.HTML`, and `mypage.Html` are all different pages.

- **Scripts traditionally are published to your CGI-bin directory.** Unless your systems administrator tells you otherwise, assume any executable pages (that is, pages that require server-side processing) should be stored in your CGI-bin directory. If you are using any canned Perl scripts, this is where you are likely to find them.
- **Permissions.** You need read, write, and execute permissions to your directories and your files. Your directories and files will also have to be set so everyone has read and execute permissions. You can check your permissions by using some FTP clients or by using Telnet, if you are authorized access to your files via Telnet.

Looking at Windows NT Servers

Windows NT servers are newer to the field, but offer their users many advantages. The biggest advantage is the interface looks just like the PC that is probably sitting on your desk. For Windows 95, Windows 98, and Windows NT users, you can drag-and-drop into a Windows NT server environment as easily as you can onto your own local hard drive. Another advantage of publishing to a Windows NT Web server is capitalization doesn't matter. Unless you *wanted* `Mypage.html` to be a different page from `MyPage.html`, this is a real advantage!

A downside of publishing to a Windows NT server is you can't check file and directory permissions remotely. This is a source of frustration for server administrators, as well. You must test thoroughly to be sure others can see your files and execute your scripts, because no one place exists where you can go to check your permissions. To make matters worse, if the permissions on a higher directory are too restrictive, you can't see a page even if the permissions for that directory and page are set appropriately. The only solution to this is to test, test, test!

From Here

Proceed to Part III to start laying the foundation of your Web pages.

Jump to Chapter 45 to begin accessing external databases

Summary

In this chapter you learned about some of the fancier things you can do with your Web server. First, you learned about the client-server model and how you can use client-side processing to reduce the load on your server. You also learned about

Web servers and permissions. Included were some tidbits about how you can use middleware to communicate with a database on your server.

You learned about how you can use cookies to do more sophisticated hit counting. Secure servers were also discussed, including the four steps you must take to implement a secure page on your site. Finally, you looked in-depth at UNIX and NT Web servers.

✦ ✦ ✦